

# Accelerating Implicit Finite Difference Schemes Using a Hardware Optimised Implementation of the Thomas Algorithm for FPGAs

Samuel Palmer<sup>\*</sup> and David Thomas<sup>\*\*</sup>

<sup>\*</sup>Department of Computer Science, University College London, Gower Street, London, WC1E 6BT

<sup>\*\*</sup>Department of Electrical Engineering, Imperial College London, Exhibition Road, London, SW7 2AZ

**Abstract**—The design and implementation of the Thomas algorithm optimised for hardware acceleration on an FPGA is presented. The hardware based algorithm combined with custom data flow and low level parallelism available in an FPGA reduces the overall complexity from  $8N$  down to  $5N$  arithmetic operations, and combined with a data streaming interface reduces memory overheads to only 2  $N$ -length vectors per  $N$ -tridiagonal system to be solved. The Thomas Core developed allows for multiple tridiagonal systems to be solved in parallel, giving potential use for solving multiple implicit finite difference schemes or accelerating higher dimensional alternating-direction-implicit schemes used in financial derivatives pricing. This paper also discusses the limitations arising from the fixed-point arithmetic used in the design and how the resultant rounding errors can be controlled to meet a specified tolerance level.

## I. INTRODUCTION

One method of pricing financial products, such as options, is to use finite difference (FD) schemes [1]. FD schemes provide a numerical method for solving partial differential equation (PDEs) for which a tractable analytical solution may not exist.

It is important that a high degree of accuracy is obtained when using FD methods, but this results in a quadratic increase in computation [2] which ultimately results in an increase in time taken. Timing may be crucial for traders wanting to analyse the real time risk of a large portfolio or the value of a trading opportunity. This provides the motivation to develop an accelerator which can allow financial products to be priced faster and more accurately. In similar work Jin et al [2] presents an FPGA based accelerator for the explicit FD scheme, though the explicit scheme has the significant drawback of being conditionally unstable which limits its accuracy and application. Instead, implicit based schemes provide unconditional stability producing reliable and accurate solutions, but has the disadvantage of being computationally more expensive than the explicit scheme due to the need to solve the tridiagonal matrix inversion problem  $Ax = y$ . This work presents the design for an FPGA based tridiagonal solver (Thomas Core) which can be used as a standalone accelerator to solve tridiagonal systems of equations, or as part of a larger FPGA based implicit solver, used to solve PDEs to a high degree of accuracy in one or many dimensions.

## A. FPGAs

Field programmable gate arrays (FPGAs) provide an integrated circuit that can be reconfigured on the fly or 'in the field'. FPGAs provides a flexible and cost effective way to develop and implement custom hardware designs. For applications in high performance computing they can result in greater acceleration and lower power consumption compared to rival technologies such as graphics processing units (GPUs) [3].

The power of FPGA acceleration lies in the ability for custom parallelism to be incorporated in the algorithm which may not be possible in other processing devices such as GPUs and CPUs. This low level parallelism allows the parallelisation of the usually-serial algorithms, which is the focus of this work. The other advantage FPGAs can offer over GPUs is data-streaming; for a GPU all the data must first be transferred onto the device before computation. On an FPGA it is possible to stream the data onto the device and begin computation (known as data flow) whilst data is still arriving. In this design we shall take advantage of data streaming capabilities to increase overall speed and reduce on-chip memory overheads.

FPGAs have been used in financial engineering to accelerate the pricing of derivatives and other products with complex mathematical pricing equations. The first wave of pricing applications focuses on applying Monte Carlo methods [4], the attraction of Monte Carlo methods being that they are very easily parallelisable given that many independent asset simulations need to be constructed and are relatively simple in terms of complexity. In comparison to GPUs, FPGA based Monte Carlo produced significant speed advantages due to the iterative structure of the algorithm [5], and as always provided significant power advantages. More recently deSchryver et al [6], have implemented multi-dimensional Monte-Carlo for the Heston PDE, which is one of the first attempts of FPGA based multi-dimensional pricing. Apart from Monte Carlo simulations other methods of derivatives pricing have been implemented, such as binomial tree, quadrature and explicit finite-difference (Ex-FD) methods. In a comparison of these methods (including Monte Carlo) on an FPGA [7], quadrature and Ex-FD were found to be the most favourable.

## B. Finite Difference Schemes and Tridiagonal Systems

Finite difference (FD) schemes are an important tool for solving parabolic partial differential equations (PDEs) numerically. In financial engineering FD methods are commonly employed to solve PDEs that are used to model derivatives, such as the famous Black-Scholes equation (BSE) [1].

Finite difference schemes begin by discretising the problem domain into a mesh/grid over the time interval  $[0, 1]$  and in basic cases, the asset price interval  $[0, S^{max}]$ . The domain is discretised into  $N$  asset price steps and  $M$  time steps, given by:

$$\Delta S = \frac{S^{max}}{N} \quad (1)$$

$$\Delta t = \frac{1}{M} \quad (2)$$

The spatial derivative terms are approximated using central difference and backward difference for the time derivative. These discretisations are then substituted into the PDE to produce the discrete difference equation, for example the BSE equation gives:

$$V_n^m = a_n V_{n-1}^{m-1} + b_n V_n^{m-1} + c_n V_{n+1}^{m-1} \quad (3)$$

with problem dependant stencil coefficient values  $a_n, b_n, c_n$ . This is the basic implicit scheme used for one dimensional problems, the resultant system of equations can be written in matrix form and needs to be solved for the price vector at the current time-step,  $V^{t-1}$ , where the vector  $V^t$  is known from the previous implicit step.

$$AV^{t-1} = V^t \quad (4)$$

or more generally written as the matrix inversion problem  $Ax = y$ , where the coefficient matrix  $A$  takes on the banded tridiagonal form shown below:

$$A = \begin{bmatrix} b_0 & c_0 & 0 & 0 & 0 & \dots \\ a_1 & b_1 & c_1 & 0 & 0 & \dots \\ 0 & a_2 & b_2 & c_2 & 0 & \dots \\ 0 & 0 & a_3 & b_3 & c_3 & 0 & \dots \\ \vdots & & & & & & \\ \vdots & & & & & & \\ \vdots & & & & & & \\ 0 & & & & \dots & 0 & a_n & b_n \end{bmatrix} \quad (5)$$

Further to this when introducing numerical schemes for pricing multidimensional derivatives, such as basket options or under stochastic volatility, another class of finite difference schemes known as alternating-direction-implicit schemes [8] may be used. These schemes solve the PDE in an implicit manner within multiple dimensions. These methods can be computationally challenging as they require the solution to multiple tridiagonal systems at each time step, thus a lot of effort has gone into creating fast parallel solvers on devices such as GPUs [9] [10].

## C. Thomas Algorithm

---

### Algorithm 1 Thomas Algorithm (a,b,c,y) Pseudo Code

---

```

d[0] = b[0]
z[0] = y[0]
for i = 1 to N do
    prev = i - 1
    li = a[i]/d[prev]
    d[i] = b[i] - li*c[prev]
    z[i] = y[i] - li*z[prev]
end for
z[N] = z[N]/d[N]
for i = N-1 to 0 do
    x[i] = (z[i] - c[i]*x[i+1])/d[i]
end for
return x[i]
```

---

The Thomas algorithm [11] is the simplest method used to solve a tridiagonal system of equations and is commonly employed on serial devices such as a CPU. The Thomas algorithm is a specialised case of gaussian elimination and can be derived from the LU decomposition of the matrix  $A$ . This reduces the system down to the solution of two bi-diagonal systems which can then be solved via gaussian elimination. The first system is solved via forward substitution and the second system is solved via backward substitution. These two stages will be referred to as the forwards and backwards iterations. The Thomas algorithm is given in algorithm 1, it has a complexity of  $O(N)$  and requires a total of  $8N$  arithmetic operations to solve an  $N$ -tridiagonal system.

In parallel computing the Thomas algorithm is usually less favoured than algorithms such as recursive-doubling [12], cyclic-reduction [13] and parallel cyclic-reduction [14], since although these algorithms have a larger number of arithmetic operations some of the operations can be parallelised on devices such as GPUs [15] resulting in an overall lower algorithmic complexity. With a recent increased interest in FPGA acceleration attempts have been made to port tridiagonal solvers onto FPGAs [16][17][18]; in this application the simplicity of the Thomas algorithm makes it well suited to the task when compared to cyclic-reduction which maybe too complex for efficient FPGA implementation [18].

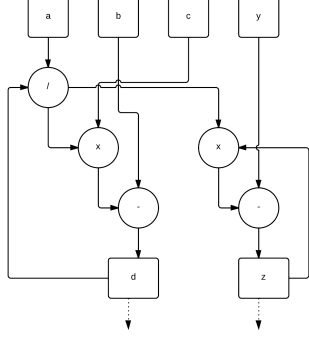


Fig. 1. Data dependency graph for the forward iteration of the Thomas algorithm

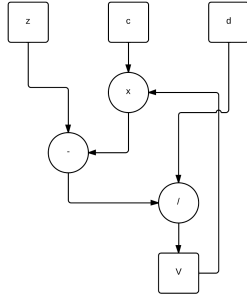


Fig. 2. Data dependency graph for the backwards iteration of the Thomas algorithm

## II. ALGORITHMIC OPTIMISATION AND LOW LEVEL PARALLELISATION

Figures 1 and 2 depict the data dependency of the Thomas algorithm; it can be observed that in the forward iteration there are two separate branches of computation, one for calculating  $d_n$  and the other for  $z_n$  and hence this provides the first level of parallelism extracted. A similar approach has been taken by both Oliveira et al [16] and Warne et al [17]. This optimisation reduces the effective serial arithmetic operations down from  $8N$  to  $6N$ .

The problem with this simple optimisation is that although there is a reduction in serial operations, it has only reduced a multiply and a subtract which are computational cheap when compared to divisions. Consequently a competitive speed-up over faster clocking devices such as CPUs may not be obtained [17]. We thus introduce a simple algorithmic rearrangement that can allow for the two divisions from the backwards and forward iterations to be effectively parallelised. Equation 6 shows the factorisation of the backwards iteration calculation where we now treat the divisions of  $z_n$  and  $c_n$  by  $d_n$

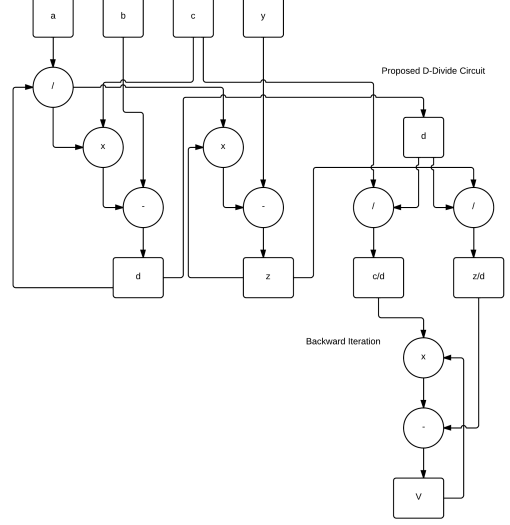


Fig. 3. Data dependency graph for the proposed Thomas algorithm structure optimised for FPGA implementation.

individually.

$$\frac{z_n - c_n V_{n+1}}{d_n} = \frac{1}{d_n} z_n - \left( \frac{1}{d_n} c_n \right) V_{n+1} \quad (6)$$

In a serial implementation this would add an extra division to the total number of arithmetic operations, which is not usually desirable, but as shown in figure 3 the data dependence is in fact here reduced, we can now treat these two divisions in parallel with each other whilst also performing them in parallel with the forward iteration calculations. This reduces the serial arithmetic operations down from the original  $8N$  to  $5N$ .

For FPGA implementations this rearranged algorithm has two advantages:

- 1) Total latency of the algorithm is almost halved by parallelising the two lengthy divisions.
- 2) Memory requirements are reduced from the need to save three intermediate data vectors ( $c$ ,  $z$  and  $d$ ) to two ( $c/d$  and  $z/d$ ).

### A. Pipelining

Further to the low level algorithmic optimisations higher level parallelism can be achieved in two ways: pipelining the data through the computation of the forward and backward iterations; and pipelining the sets of data between the forward and backward iterations, which has commonly been implemented for multiple CPU versions to parallelise the Thomas algorithm [19].

Firstly the computational units themselves can be deeply pipelined, an approach used by Olivera et al [16], which allows for multiple independent tridiagonal systems to be computed in the same iteration cycle. For example, if the forward iteration computational unit has  $P_F$  pipeline stages then throughout one iteration it is possible to fill each stage of the pipeline with a

computation allowing for  $P_F$  independent tridiagonal systems to be computed.

The second type of pipelining of the algorithm means that given a set  $T$  of pipelined tridiagonal systems for either iteration, as discussed above, we can simultaneously compute the forward and backwards iterations of the two different sets (given the first set has already been through the forward iterations) independently in one Thomas Core. Thus at maximum throughput the Thomas Core can effectively calculate solutions for  $2T$  independent systems.

### B. Design Latency and Throughput

The first simple bound for the size of  $T$ , the number of pipeline tridiagonal systems for each iteration stage, is given by the limited size of the forward iteration pipeline stages  $P_F$ :

$$T \leq P_F \quad (7)$$

This is given that the pipeline stage in the backward iteration,  $P_B$ , satisfies  $P_B \leq P_F$ . In the case that  $P_B < P_F$  simple replication of the backwards iteration computational units will allow the computation of  $P_F$  systems.

A further limitation is due to the data rate, for which  $r$  denotes the time required to receive a row of data, and  $r_{sys}$  the time to send  $N$  rows. Given that the system is data-driven, stalling will occur when awaiting for new data to be streamed when it is required for computation in the forward iteration therefore certain considerations must be made in order to achieve maximal performance.

We shall first define variables used in the proceeding calculations.

- $L_F$  and  $L_B$  are the latency for a single forwards and backwards iteration respectively.
- $L_{DDiv}$  is the latency for the  $\frac{z_n}{d_n}$  and  $\frac{c_n}{d_n}$  computation.
- $t_r$  is the time taken for a one row of data, that is  $a_n, b_n, c_n$  and  $y_n$  to be transferred to the FPGA.
- $t_{sys}$  is the time taken to send a whole N-tridiagonal system,  $t_{sys} = Nt_r$ .
- $t_{result}$  is the time taken to send back all  $N$  results.
- $f$  is the clock frequency of the hardware.

The latency for each of the calculation stages can be found from the number of pipeline stages

$$L_F = \frac{P_F}{f} \quad (8)$$

$$L_B = \frac{P_B}{f} \quad (9)$$

Hence from this the time,  $t$ , to compute one N-tridiagonal system is given by:

$$t = NL_F + NL_B + L_{DDIV} \quad (10)$$

We shall now examine how to obtain maximal performance for the Thomas Core with respect to data transfer times. The simplest case is when  $t_r \geq L_F$  i.e. when the row data time is less than the latency of the forward iteration. In this case  $T = 1$ , otherwise time which could have been used in computation

is wasted when the computation is stalled waiting for the next tridiagonal systems row to arrive. The waiting time can be defined as:

$$W_t = Tt_r - L_F \quad (11)$$

Given that the system is pipelined with  $P_F$  stages through the forward iteration  $L_F$  remains constant for up to  $P_F$  systems to be computed. But given that additional data transfers are then required if  $T > 1$  then  $W_t$  grows linearly. As such a better strategy is to wait for the core to finish computing the first system in the forward iteration, and then allow for the forward iteration of the next system to be computed in parallel whilst the first is then backwards iterating. In this case the minimum latency to calculate the solution to two N-tridiagonal systems is:

$$2t = 2t_{sys} + 2t_{result} + NL_B + L_{DDiv} + L_F \quad (12)$$

Given that  $L_B$  is suitably small and in combination with a reasonable size of  $N$  then negligible computational latency is expected in comparison to data transfer times. Hence the computational latency per tridiagonal system is bounded by the speed at which you can get the data onto and off of the device

$$t > t_{sys} + t_{result} \quad (13)$$

in the case of a GPU based solver this can not be possible and computational latency is always in addition to the data transfer latency. Hence under the scenario for which the same transfer latency is achieved on both the FPGA and GPU, the FPGA will always provide greater acceleration.

The more interesting case is when  $r \geq L_F$ , this firstly implies that we have a negative waiting time,  $W_t$ , which means that the row data is ready before it is needed for computation. Therefore it is possible to fill the pipeline  $P_F$ . Here  $T$  is limited by the minimum number of complete rows that can be received during during one forward iteration.

$$T = \text{floor}\left(\frac{L_F}{t_r}\right) \quad (14)$$

In this case the overall latency for solving a maximum of  $2T$  systems is given by

$$2Tt = t_r + 2Tt_{result} + NL_F + NL_B + L_{DDiv} \quad (15)$$

The average latency per system is bounded by the performance of the Thomas Core where:

$$t > t_{result} + \frac{NL_F + NL_B}{2T} \quad (16)$$

Although computationally the system can theoretically handle  $T = P_F$ , a stronger bound can be placed on  $T$  in terms of memory resources available to the system. During the Thomas algorithm it is required that 2 data vectors are stored in memory from the forward iterations to be used in the backward iterations. This therefore gives a memory requirement,  $M$ :

$$M = 2TNB \quad (17)$$

where  $B$  is the data width, which in the prototype system is 32 bits or 4 bytes. For this system with a maximum of  $T = P_F = 60$  and  $N_{max} = 1024$  the memory requirement

is 480KB, which can fit onto the block ram available on the Zynq-7020. Using the original algorithm where 3 vectors are required for storage, this would not fit onto the Zynq-7020. Therefore in this case we can either run the Thomas Core in parallel for a singular set of  $T = 60$  or two pipelined sets where  $T_1 = T_2$  at just over 30 systems per set, for these large theoretical values of  $T$  the size of the sets may instead be limited by the data transfer rate rather than the memory overheads.

### III. IMPLEMENTATION

The system has been prototyped on the Zedboard development board which uses a Zynq-7020 system-on-chip (SoC). The Zynq-7020 combines two ARM Cortex-A9 CPUs with an Artix-7 Xilinx FPGA and an array of other peripherals onto a single chip. The Zynqs' CPUs interface with the Xilinx Artix-7 FPGA via high speed AXI4 interconnects and provide a simple way for prototyping heterogeneous computing systems.

The previously discussed optimised Thomas algorithm was here implemented in VHDL-93 using LogiCore IP Cores [20] for the fixed point arithmetic units. Fixed point arithmetic has been chosen due to its significantly faster speed compared to floating point arithmetic units, adding to the FPGAs' advantage. As will be discussed later this does result in finite precision errors but which can be controlled to a given tolerance.

As previously mentioned the Thomas Core will be a data driven system which allows it to harness the data streaming capabilities of the FPGA; unlike a GPU which would require all the data to be transferred onto the device even before computation begins, here computation can begin as soon as a row of data is received. Thus in the event of receiving new data the computation is activated. The Thomas Core itself has been extended to include FIFO buffers for receiving data and transmitting results between the CPU and floating/fixed point conversion units. The use of FIFO buffers allows the data driven system to cope with situations where the streamed tridiagonal row data is either slower or faster than the rate of computation of the Thomas Core, whilst the conversion units allow the floating point CPU data to be transferred to the core (and results back to the CPU) without adding further latency of conversion code in the software driver.

The current prototype system uses the FPGA resources shown in table I, where the design has been constructed with a focus on speed rather than area of the FPGA used. The design can be clocked up to 200-250MHz. It should be noted that the current prototype does not implement all of the features proposed in the this paper, but they are possible to be integrated in a production version of the core.

TABLE I  
FPGA RESOURCES USED IN THE IMPLEMENTED DESIGN

#### A. Implicit Finite-Difference Accelerator

To implement the Thomas Core in an accelerated FD solver the main FD algorithm will be hosted on the Zynqs' CPU

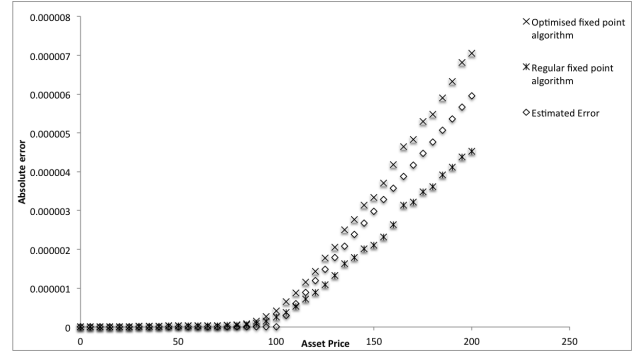


Fig. 4. Absolute error for the fixed point Thomas algorithm using a 24 bit fractional representation relative to floating point arithmetic.

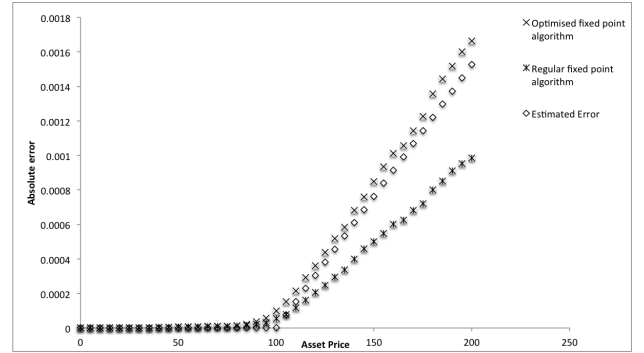


Fig. 5. Absolute error for the fixed point Thomas algorithm using a 16 bit fractional representation relative to floating point arithmetic.

(single core). The CPU will be responsible for three stages of the calculation: calculating the stencil coefficients; writing the coefficients and  $y_n$  to the FPGA; and finally receiving the results back from the FPGA. The advantage of the using the Thomas Core when implementing the FD schemes is that whilst the row  $n$  data is being used on the FPGA, the new coefficients at  $n + 1$  can be computed in parallel with the Thomas Core computations on the CPU. Hence this further reduces the overall complexity and time for large calculations, meaning that the CPU only needs to perform  $3N$  steps to calculate each coefficient and the  $4N$  transfers of the row data which replaces the memory write CPU instructions. Thus theoretically at least a  $8N$  speed-up should be observed for the overall scheme. Further to this we can enhance calculation times by using some optimisation signals. If the *loop-opt* flag is asserted then instead of the data being written back to the CPU it is written straight out of the results FIFO into the  $y$  data-in FIFO. This can be done whilst the CPU writes in the new stencil coefficients and the calculation proceeds as before.

### IV. SIMULATED RESULTS

#### A. Finite Precision

Due to the finite precision effects of the fixed point numerical representation used for the arithmetic it is expected that there will be a loss in the degree of precision of the results when compared to floating point arithmetic solvers. In a fixed point representation be real number is represented as a binary

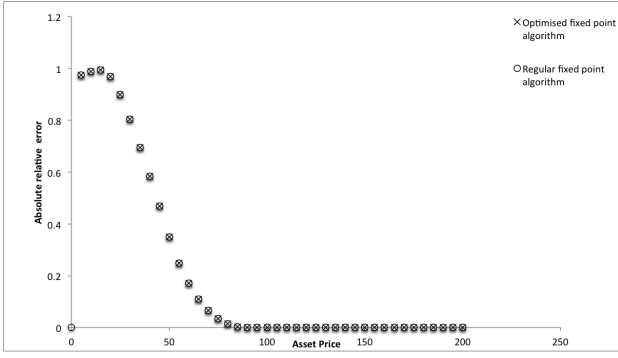


Fig. 6. Absolute relative error for the fixed point Thomas algorithm using a 24 bit fractional representation relative to floating point arithmetic.

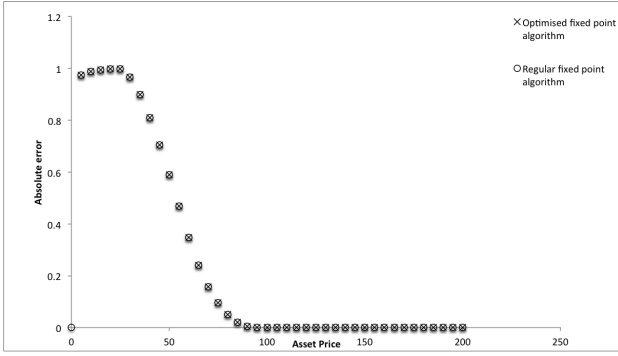


Fig. 7. Absolute relative error for the fixed point Thomas algorithm using a 16 bit fractional representation relative to floating point arithmetic.

integer, where  $int$  bits are allocated to represent the integer part of the number and the remaining  $frac$  bits are used to represent the fractional. Here, 24 and 16 fractional bits have been used. It has been assumed that there is a sufficient number of bits for the integer representation and overflow is not a concern.

The precision of the optimised algorithm has been tested via fixed point simulations against the floating point Thomas algorithm and the regular (non optimised) fixed point Thomas algorithm. In these simulations 1,000,000 tridiagonal systems have been solved, these tridiagonal systems have been created from a set of random implicit finite difference derivative pricing problems; the parameters for the pricing problem set are:

- $S^{max} = 200$
- $N = 20$
- $M = 1000$
- $Strike = 100$
- $r \sim U[0.01, 0.06]$
- $\sigma \sim U[0.10, 0.30]$

where  $r$  and  $\sigma$  are drawn from uniform distributions, and should be indicative of pricing parameters that the solver maybe applied to. As such 10000 random problems are generated with  $dt = 0.001$  resulting in the 1,000,000 tridiagonal systems to be solved. The absolute error,  $|e_n|$ , for each price of the fixed point algorithms,  $x_n$ , has then been calculated with

respect to the floating point algorithm price  $x_n^{fl}$ :

$$|e_n| = |x_n - x_n^{fl}| \quad (18)$$

The absolute relative error,  $|e_n^{rel}|$ , for each price of the fixed point algorithms,  $x_n$ , has also been calculated:

$$|e_n^{rel}| = \frac{|e_n|}{|x_n^{fl}|} \quad (19)$$

The absolute errors for 24 and 16 bit fractional representations are shown in figures 4 and 5, and the absolute relative errors in figures 6 and 7 respectively. Apart from the price at  $n\Delta S = 0$ , it can be seen that there is always a finite precision error relative to floating point arithmetic. For the regular and optimised fixed point algorithms it can be seen that the relative error tends to 1 as the floating point result tends towards the finite precision limit of the fixed point representation and thus the fixed point result tends to 0. Comparing the precision of the two fixed point algorithms, as expected, the extra arithmetic operations in the optimised algorithm does result in a small additional error, though respectively there is very little significant difference in the error between the two algorithms.

As an approximation to the magnitude of error it can be observed that the absolute error function is a similar shape to the option price curve. Hence we can assume the error takes the form of a linear function of the discounted expected payoff (i.e the price of an option):

$$|e_n| \approx Ke^{-rt}\mathbb{E}[\text{Payoff}(n\Delta S)] \quad (20)$$

it can also be assumed that the error is a linear function of the expected round off error of the finite precision representation with fractional bits,  $frac$ . Here we shall also assume that the round-off error is uniformly distributed, hence in the case of round-half-up rounding the expectation of the roundoff error is:

$$K = 0.5^{frac} \quad (21)$$

from this, bounds for the option price at a given underlying asset price,  $B(n\Delta S)$ , such as the non-parametric upper bound by Lo[21], can be used as an estimate of the final option price. As shown in figures 4 and 5 this provides a good estimate for the magnitude of the absolute error for in-the-money options. When  $n = N$  the maximum absolute error,  $|e_n^{max}|$ , of the solver is given by:

$$|e_n^{max}| \approx KB(N\Delta S) \quad (22)$$

Hence the absolute relative error can be approximated as  $K$ . Though it must be stressed that these relationships only hold for in-the-money options, on closer inspection of  $n\Delta S < Strike$ , the relationship is far from linear suggesting a different type of behaviour. This suggests then that the respectively large final price of in-the-money options dominates smaller error effects related to the iterative propagation within the algorithm which are more dominant when the price is small and tends towards 0. These small effects will be related to the roundoff errors of the tridiagonal coefficients and intermediate calculated values. Rearranging equation 22 with respect to  $frac$  we can determine the minimum required fractional bits

so that the system can meet a given maximum absolute error tolerance.

$$frac \geq \frac{\ln |e^{max}| - \ln B(N\Delta S)}{\ln 0.5} - 1 \quad (23)$$

Although an exact relationship for the absolute error for out-of-the money options is not given here, a weak condition for bounding the maximum relative error can still be posed, for prices above a certain  $n$  we must ensure that:

$$B(n\Delta S) > K \quad (24)$$

and as such:

$$|e_n^{rel}| \approx 1 \quad ; B(n\Delta S) < K \quad (25)$$

i.e. there needs to be enough fractional bits to suitably represent the price at  $n$ , otherwise  $|e_n| \rightarrow x_{fl}$  and thus  $e_n^{rel} \rightarrow 1$ . Giving the second condition for  $frac$ :

$$frac \geq \frac{\ln B(n\Delta S)}{\ln 0.5} - 1 \quad (26)$$

When applied to the whole implicit pricing problem the maximum absolute error for in-the-money options exhibits a similar linear relationship with the final price but is amplified by the number of iterations, i.e.  $\frac{1}{\Delta t}$ . For implicit pricing  $K$  is now:

$$K = \frac{1}{\Delta t} 0.5^{frac+1} \quad (27)$$

which gives the minimum number of fractional bits with respect the maximum absolute error as:

$$frac \geq \frac{\ln \Delta t |e^{max}| - \ln B(N\Delta S)}{\ln 0.5} - 1 \quad (28)$$

The equations given above are derived via qualitative arguments specific to the given problem, further work will look at the effects of finite precision roundoff errors as a function of the matrix coefficient values and righthand side vector, using perturbation theory to provide a general optimisation framework for any given tridiagonal system and error tolerance. This will then allow more accurate optimisations of the finite precision data-width with respect to speed, memory capacity and required error tolerance for the problem.

### B. Acceleration

Figure 8 presents the theoretical computational latency (excluding data transfer time) for the proposed Thomas Core for solving a single  $N$ -tridiagonal system, using equation 10 with clock frequency 200MHz (preliminary timing analysis shows the design can be clocked 200-250MHz). It can be seen that when compared to the CPU implementation of the Thomas algorithm and the best performing GPU solver taken from [15] the Thomas Core proposed in the work should have similar performance to the GPU for small problem sizes and then greater performance for large problem sizes. Taking into consideration data transfer times, if equal for the GPU and FPGA, the FPGA will provide further acceleration. This is also not taking into consideration that this solver has the ability to process multiple tridiagonal systems at maximum throughput.

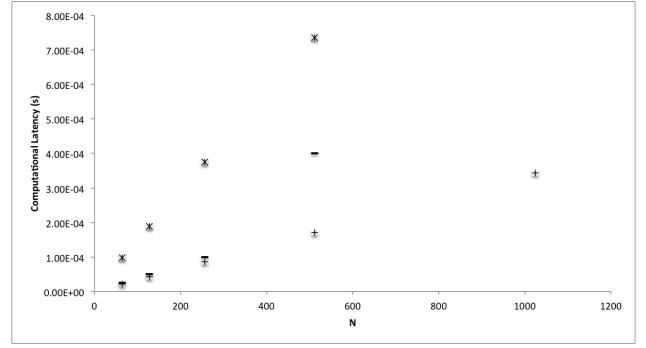


Fig. 8. Theoretical computational latency for the propose Thomas Core compared to the best parallel GPU solver by Zhang et al[15] and python desktop CPU implementation of the serial Thomas algorithm, for different problems sizes,  $N$ . Legend : \* CPU @2.6GHz, - Best GPU, + FPGA Optimised Thomas Algorithm @200MHz.

## V. IMPLEMENTATION RESULTS

Implementation results are still pending due to an unforeseen issue with the vendor hardware compiler for the device.

### A. Tridiagonal Solver

### B. Implicit Finite-Difference Acceleration

The presented solver will now be used as part of a full implicit option pricer, two modes of operation will be tested as described in the implementation section: where each individual tridiagonal system is transferred to and from the FPGA, and where the *loop-opt* signal is used to provide reduced data transfer requirements by retaining data on the FPGA.

## VI. CONCLUSIONS AND FUTURE WORK

The greatest limitation of this work is not the solver itself but the latency required to interact with the solver at maximum throughput. Here we have only tested the solver when plugged into a heterogeneous system with a sub-maximal data rate. For the solver to be useful in a heterogeneous system the data transfer time,  $t_D$ , for transferring the all the rows to the system and the receiving the result vector, must be less than the serial CPU Thomas algorithm otherwise its use is obsolete.

Providing an efficient parallel tridiagonal solver on the FPGA fabric does now open up the opportunity to apply implicit based finite-difference methods onto FPGAs, and further to this, alternating-direction-implicit methods which can fully utilise the parallelism offered by the Thomas Core.

Future work will look at implementing the Thomas Core on a PCI-express based FPGA board, interacting as a specialised accelerator for a desktop computer. This work will also be extended by migrating the implicit finite difference schemes onto the FPGA to provide a massively parallel coarse grain implicit solver.

## REFERENCES

- [1] D. Tavella and C. Randall, *Pricing financial instruments: The finite difference method*. John Wiley & Sons New York, 2000.
- [2] Q. Jin, D. Thomas, and W. Luk, "Exploring reconfigurable architectures for explicit finite difference option pricing models," in *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, pp. 73–78, 2009.
- [3] S. Che, J. Li, J. Sheaffer, K. Skadron, and J. Lach, "Accelerating compute-intensive applications with gpus and fpgas," in *Application Specific Processors, 2008. SASP 2008. Symposium on*, pp. 101–107, 2008.
- [4] P. Glasserman, *Monte Carlo Methods in Financial Engineering (Stochastic Modelling and Applied Probability)* (v. 53). Springer, 1 ed., Aug. 2003.
- [5] X. Tian and K. Benkrid, "High-performance quasi-monte carlo financial simulation: Fpga vs. gpp vs. gpu," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 3, no. 4, p. 26, 2010.
- [6] C. de Schryver, P. Torruella, and N. Wehn, "A multi-level monte carlo fpga accelerator for option pricing in the heston model," in *Proceedings of the Conference on Design, Automation and Test in Europe, DATE '13*, (San Jose, CA, USA), pp. 248–253, EDA Consortium, 2013.
- [7] Q. Jin, W. Luk, and D. Thomas, "On comparing financial option price solvers on fpga," in *Field-Programmable Custom Computing Machines (FCCM), 2011 IEEE 19th Annual International Symposium on*, pp. 89–92, 2011.
- [8] D. Peaceman and H. Rachford, Jr., "The numerical solution of parabolic and elliptic differential equations," *Journal of the Society for Industrial and Applied Mathematics*, vol. 3, no. 1, pp. 28–41, 1955.
- [9] D. M. Dang, C. Christara, and K. Jackson, "A parallel implementation on gpus of adi finite difference methods for parabolic pdes with applications in finance," *Available at SSRN 1580057*, 2010.
- [10] D. Egloff, "Gpus in financial computing part iii: Adi solvers on gpus with application to stochastic volatility," *Wilmott mag.*, March, pp. 51–53, 2011.
- [11] L. Thomas, "Elliptic problems in linear differential equations over a network," *Watson Sci. Lab Report. Columbia University, New York*, 1949.
- [12] H. S. Stone, "An efficient parallel algorithm for the solution of a tridiagonal linear system of equations," *J. ACM*, vol. 20, pp. 27–38, Jan. 1973.
- [13] R. W. Hockney, "A fast direct solution of poisson's equation using fourier analysis," *J. ACM*, vol. 12, pp. 95–113, Jan. 1965.
- [14] R.W.Hockney and C.R.Jesshope, *Parallel Computers*. Adam Hilger, 1981.
- [15] Y. Zhang, J. Cohen, and J. D. Owens, "Fast tridiagonal solvers on the gpu," *ACM Sigplan Notices*, vol. 45, no. 5, pp. 127–136, 2010.
- [16] F. Oliveira, C. Santos, F. Castro, and J. Alves, "A custom processor for a tdma solver in a cfd application," in *Reconfigurable Computing: Architectures, Tools and Applications* (R. Woods, K. Compton, C. Bouganis, and P. Diniz, eds.), vol. 4943 of *Lecture Notes in Computer Science*, pp. 63–74, Springer Berlin Heidelberg, 2008.
- [17] D. Warne, N. A. Kelson, and R. F. Hayward, "Solving tri-diagonal linear systems using field programmable gate arrays," 2012.
- [18] G. Chatziparaskevas, B. Kienhuis, and J. Walters, "An fpga-based parallel processor for black-scholes option pricing using finite differences schemes," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pp. 709–714, 2012.
- [19] I. S. Duff and H. A. van der Vorst, "Developments and trends in the parallel solution of linear systems," *Parallel Computing*, vol. 25, no. 134, pp. 1931 – 1970, 1999.
- [20] Xilinx, "Xilinx core generator," *Xilinx User Guides*.
- [21] A. W. Lo, "Semi-parametric upper bounds for option prices and expected payoffs," *Journal of Financial Economics*, vol. 19, no. 2, pp. 373–387, 1987.